1. 实验目标

1.融会贯通本课程各章节的内容，通过知识的综合运用，加深对计算机系统各功能部件的工作原理及相互联系的认识，加深计算机工作中“时间-空间”概念的理解，从而清晰地建立计算机的整机概念；

2.学习设计和调试计算机的基本步骤和方法，提高使用Quartus等软件仿真工具和集成电路的基本技能；

3.培养科学研究的独立工作能力，取得工程设计与组装调试的实践和经验；

4.该项目目的是设计一个简单的CPU（中央处理器），该CPU具有基本指令集，我们将利用其指令集生成一个非常简单的程序来验证其性能。

1. 实验原理

以控制器为中心，首控制器从指令寄存器取得指令，编译指令，再输出微控制信号，控制ALU的运算，PC加一，并且从RAM中取出数据运算，运算后再把结果通过数据总线存到RAM，在指令寄存器读去下一条指令，依次循环。

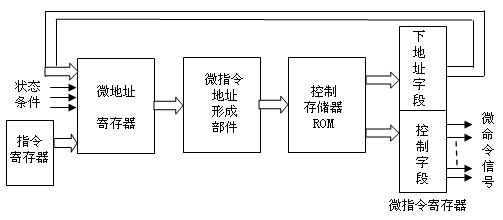
1. 模型机设计要求

|  |  |
| --- | --- |
| 总线 | 单总线结构 |
| 数据线、地址线 | 8位 |
| 指令系统 | 寻址方式  2种（立即数寻址、直接寻址）  类型    5种（算术运算、逻辑运算、数据传送、程序控制）  指令 12条(add、sub、mul、div、neg、and 、not、 or、load、store、branch、halt) |
| 运算器 | 单累加器结构, 8种运算 |
| 控制器 | 硬布线，微控制信号根据需要确定 |

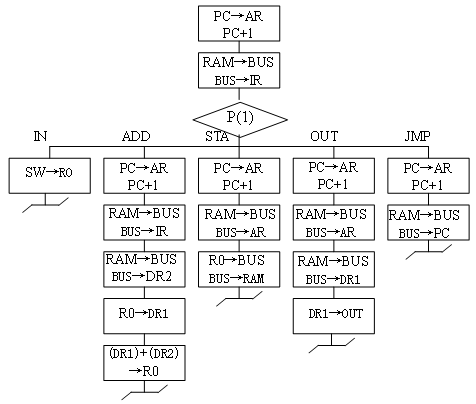
1. 模型机逻辑框架



1. 微程序控制的基本原理



1. 微程序流程图



1. 数据格式与指令系统

（1）数据格式：

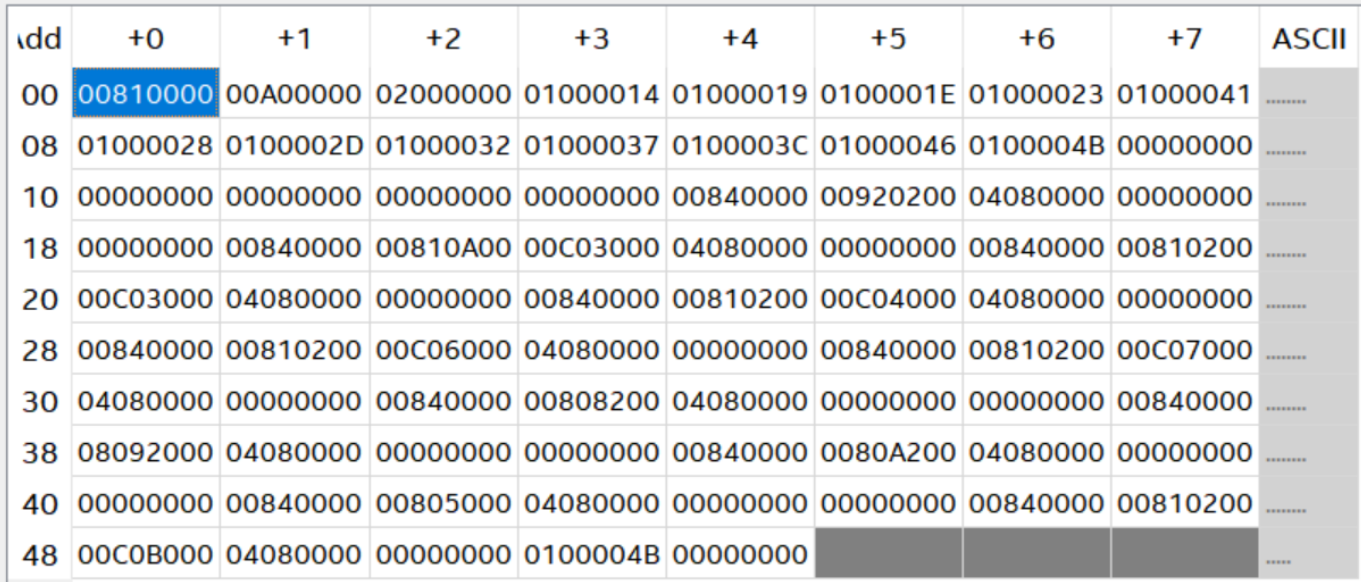
   数据的长度为8位，数据采用补码格式，相对于十进制数范围是:-27<N<27-1

（2）指令格式

|  |  |  |
| --- | --- | --- |
| **指令** | **操作码** | **功能** |
| **STORE X** | 01H | ACC→[X] |
| **LOAD X** | 02H | [X]→ACC |
| **ADD X** | 03H | ACC+[X]→ACC |
| **SUB X** | 04H | ACC-[X]→ACC |
| **JMPGZ X** | 05H | IF ACC>0 THEN X→PC ELSE PC+1→PC |
| **AND X** | 06H | ACC and [X]→ACC |
| **OR X** | 07H | ACC or [X]→ACC |
| **NOT X** | 08H | Not [X]→ACC |
| **SHIFTR X** | 09H | SHIFL ACC to RIGHT 1 bit, Logic Shift |
| **SHIFTL X** | 0AH | SHIFT ACC to LEFT 1 bit, Logic Shift |
| **MPY X** | 0BH | ACC×[X]→ACC |
| **HALT** | 0CH | HALT A PROGRAM |

1. 定做初始化数据文件

rom.mif



1. 测试数据

序号 指令字 汇编指令 说明

1 ram(0) <= x"14"; STORE 4 把累加器中的内容存入4号单元中

2 ram(1) <= x"30"; SUB 0 把累加器中的内容减去0号单元的内容，结果放在累加器中

3 ram(2) <= x"25"; ADD 5 把累加器中的内容加上5号单元的内容，结果放在累加器中

4 ram(3) <= x"15"; STORE 5 把累加器中的内容存入5号单元中

5 ram(4) <= x"46"; MUL 6 因上面操作将4号单元内容改变，为 00，即将0号单元中的值放入累加器中

6 ram(5) <= x"31"; SUB 1 因上面操作将5号单元内容改变，为 1D，即将累加器中的内容放入13号单元中

7 ram(6) <= x"55"; DIV 5 把累加器中的内容除以5号单元的内容，结果放在累加器中

8 ram(7) <= x"06"; LOAD 6 将6号单元中的值放入累加器中

9 ram(8) <= x"01"; LOAD 1 将1号单元中的值放入累加器中

1. 各功能部件VHDL的实现
2. **ALU**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity ALU is

port( clk,reset,ACCclear:in std\_logic;

     aluCONTR :in std\_logic\_vector(3 downto 0);

     BR       :in std\_logic\_vector(15 downto 0);

     PCjmp    :out std\_logic;

     ACC      :buffer std\_logic\_vector(31 downto 0));

end ALU;

architecture behave of ALU is

begin

  process(clk)

  begin

    if(clk'event and clk='0')then

     if

reset='0' then ACC<=x"00000000";

     else if

ACCclear='1' then  ACC<=x"00000000";

end if;

       if aluCONTR="0011" then   ACC<=BR+ACC;

end if;      --ADD

       if aluCONTR="0100" then   ACC<=ACC-BR;

end if;      --SUB

       if aluCONTR="0110" then   ACC<=ACC and BR;

end if;      --AND

       if aluCONTR="0111" then   ACC<=ACC or BR;

end if;       --OR

       if aluCONTR="1000" then   ACC<=not ACC;

end if;       --NOT

       if aluCONTR="1001" then                                   --SRR

          ACC(14 downto 0)<=ACC(15 downto 1);    ACC(15)<='0';

       end if;

       if aluCONTR="1010" then                                   --SRL

          ACC(15 downto 1)<=ACC(14 downto 0);     ACC(0)<='0';

       end if;

       if aluCONTR="1011" then   ACC<=ACC(15 downto 0)\*BR;       end if;     --MPY

      end if;

     end if;

     if ACC>0 then PCjmp<='1';

else

PCjmp<='0';

     end if;

  end process;

end behave;

1. **CONTROLR**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity CONTROLR is

port(

     control :in std\_logic\_vector(31 downto 0);

     R,W, RW, PCc1,PCinc,PCc3:out std\_logic;

     ACCclear,MBR\_MARc,PC\_MARc:out std\_logic;

     ACC\_MBRc,MBR\_OPc,MBR\_BRc:out std\_logic;

     CONTRout:out std\_logic\_vector(3 downto 0);

     CARc  :out std\_logic\_vector(3 downto 0);

     CAR   :out std\_logic\_vector(7 downto 0));

end CONTROLR;

architecture behave of CONTROLR is

begin

  process(control)

  begin

       CAR<=control(7 downto 0);

       PCc1<=control(8);

       PCinc<=control(9);

        PCc3<=control(10);

       ACCclear<=control(11);

       CONTRout<=control(15 downto 12);

       R<=control(16);

       W<=control(17);

       MBR\_MARc<=control(18);

       PC\_MARc<=control(19);

       ACC\_MBRc<=control(20);

       MBR\_OPc<=control(21);

       MBR\_BRc<=control(22);

       CARc<=control(26 downto 23);

       RW<=control(17);

end process;

end behave;

1. **CAR**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity CAR is

port( clk,reset :in std\_logic;

     CARc  :in std\_logic\_vector(3 downto 0);

     CAR,OP   :in std\_logic\_vector(7 downto 0);

     CARout:buffer std\_logic\_vector(7 downto 0));

end CAR;

architecture behave of CAR is

begin

  process(clk)

  begin

    if(clk'event and clk='1')then

     if reset='1' then

      if CARc="1000" then         CARout<="00000000";       end if;

      if CARc="0100" then         CARout<=OP+CARout;      end if;

      if CARc="0010" then       CARout<=CAR;              end if;

      if CARc="0001" then       CARout<=CARout+1;         end if;

     else CARout<="00000000";

     end if;

    end if;

 end process;

end behave;

1. **IR**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity IR is

port( opcode  :in std\_logic\_vector(7 downto 0);

     IRout   :out std\_logic\_vector(7 downto 0));

end IR;

architecture behave of IR is

begin

    IRout<=opcode;

end behave;

1. **PC**

保持在程序中使用的指令的跟踪

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity PC is

port( clk,PCjmp,PCc1,PCinc,PCc3,reset:in std\_logic;

     CONTRalu  :in std\_logic\_vector(3 downto 0);

     MBR\_PC   :in std\_logic\_vector(7 downto 0);

     PCout      :buffer std\_logic\_vector(7 downto 0));

end PC;

architecture behave of PC is

begin

  process(clk)

  begin

    if(clk'event and clk='0')then

      if reset='1' then

         if CONTRalu="0101" then

           if PCjmp='1' then            PCout<=MBR\_PC;

            elsif PCjmp='0' then PCout<=PCout+1;

           end if;

         end if;

         if PCc1='1' then          PCout<="00000000";       end if;

 if PCinc='1' then          PCout<=PCout+1;       end if;

 if PCc3='1' then          PCout<=MBR\_PC;       end if;

else PCout<=”00000000”

end if

end if;

  end process;

end behave;

1. **MAR**

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity MAR is

port( clk,PC\_MARc,MBR\_MARc:in std\_logic;

     PC,MBR\_MAR:in std\_logic\_vector(7 downto 0);

     MARout:out std\_logic\_vector(7 downto 0));

end MAR;

architecture behave of MAR is

begin

  process(clk)

  begin

    if(clk'event and clk='1')then

       if PC\_MARc='1' then      MARout<=PC;           end if;

       if MBR\_MARc='1' then    MARout<=MBR\_MAR;   end if;

    end if;

  end process;

end behave;

1. **BR**

ALU的输入，它保持ALU的其他操作数。

library ieee;

use ieee.std\_logic\_1164.all;

entity BR is

port( MBR\_BRc:in std\_logic;

     MBR\_BR:in std\_logic\_vector(15 downto 0);

     BRout:out std\_logic\_vector(15 downto 0));

end BR;

architecture behave of BR is

begin

  process(MBR\_BRc)

  begin

       if MBR\_BRc='1' then     BRout<=MBR\_BR;     end if;

  end process;

end behave;

1. **MBR**

包含要存储在内存中的值或从内存中读取的最后一个值

library ieee;

use ieee.std\_logic\_1164.all;

use ieee.std\_logic\_unsigned.all;

entity MBR is

port( clk, reset, MBR\_OPc, ACC\_MBRc,R,W:in std\_logic;

     ACC\_MBR :in std\_logic\_vector(15 downto 0);

     RAM\_MBR :in std\_logic\_vector(15 downto 0);

     MBR\_RAM :out std\_logic\_vector(15 downto 0);

     MBR\_BR  :out std\_logic\_vector(15 downto 0);

     MBR\_OP  :out std\_logic\_vector(7 downto 0);

     MBR\_MAR :out std\_logic\_vector(7 downto 0);

     MBR\_PC  :out std\_logic\_vector(7 downto 0));

end MBR;

architecture behave of MBR is

begin

  process(clk)

  variable temp:std\_logic\_vector(15 downto 0);

  begin

    if(clk'event and clk='0')then

     if reset='1' then

       if ACC\_MBRc='1' then  temp:=ACC\_MBR;   end if;

       if R='1' then    MBR\_BR<=RAM\_MBR;     end if;

       if W='1' then      MBR\_RAM<=temp;      end if;

       MBR\_MAR<=RAM\_MBR(7 downto 0);

       MBR\_PC<=RAM\_MBR(7 downto 0);

       if MBR\_OPc='1' then   MBR\_OP<=RAM\_MBR(15 downto 8);    end if;

      else MBR\_BR<=x"0000";

           MBR\_MAR<="00000000";

           MBR\_OP<="00000000";

           MBR\_PC<="00000000";

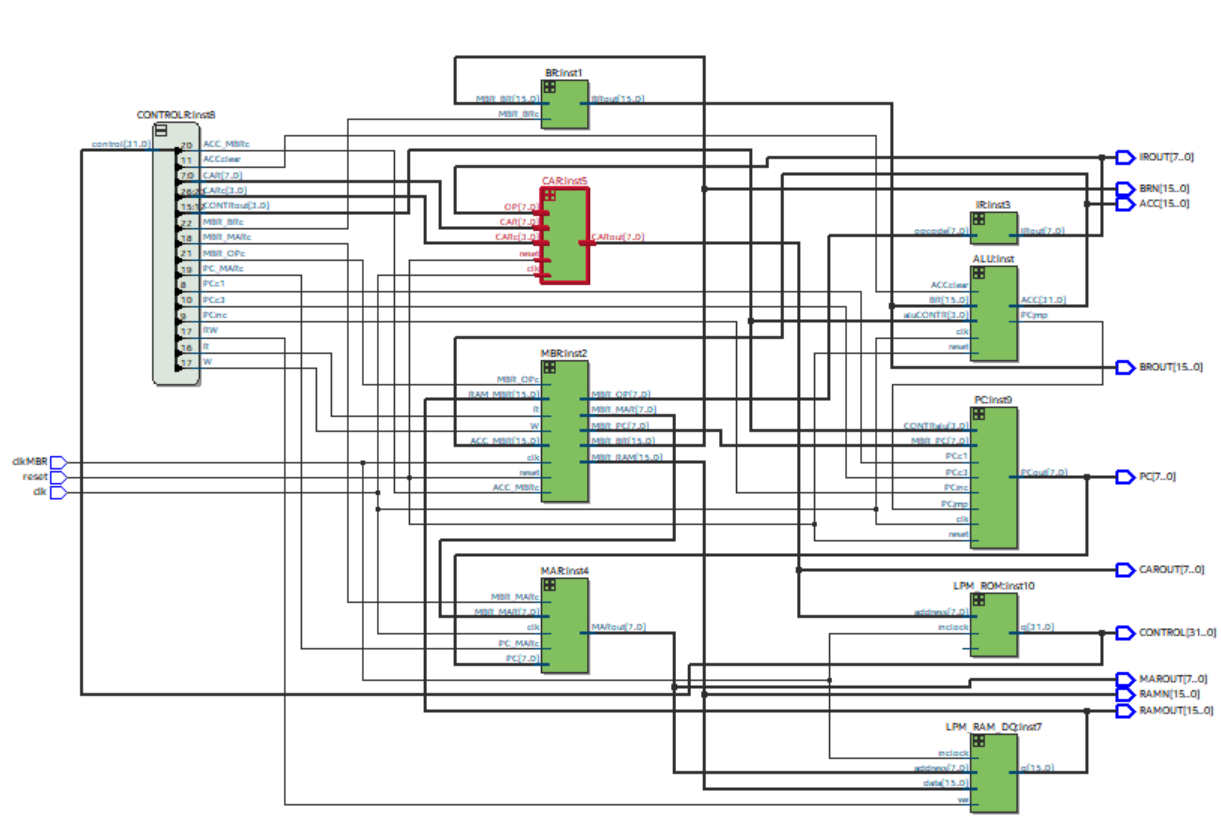
      end if;

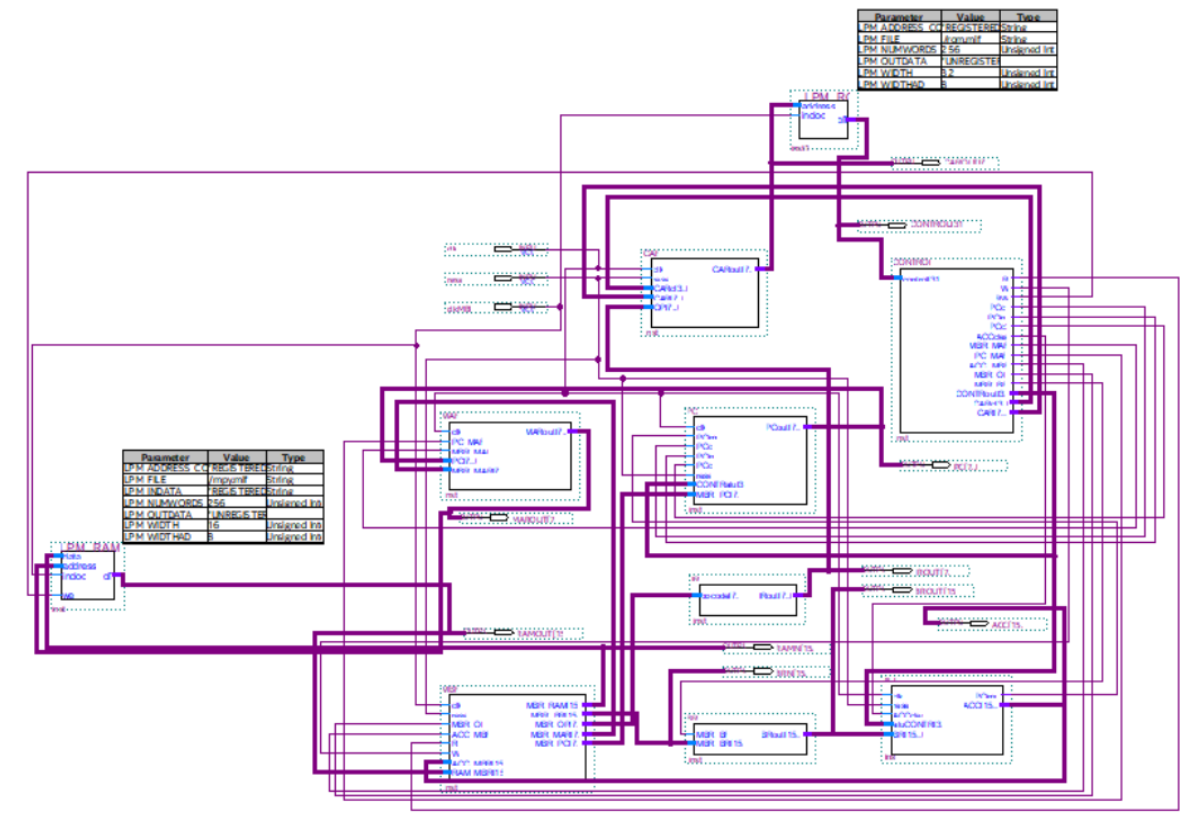
    end if;

  end process;

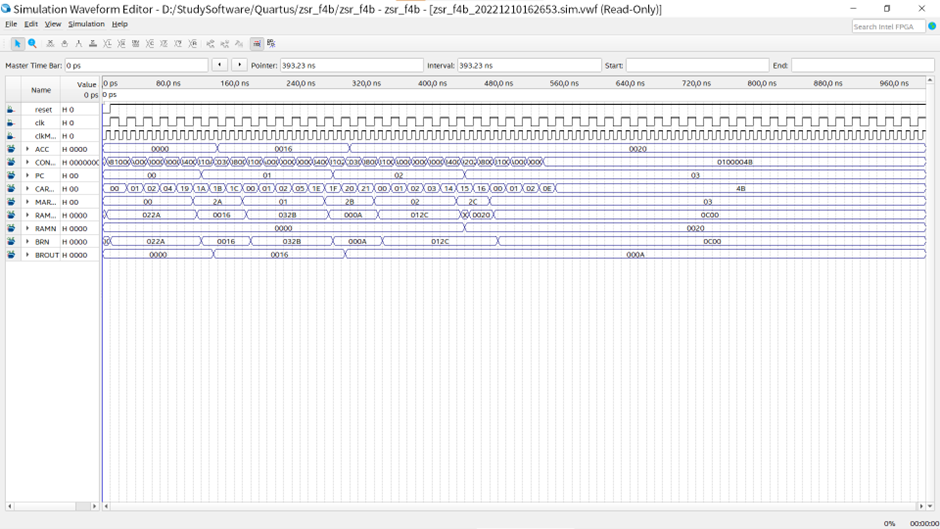
end behave;

1. 实验过程
2. 8位CPU电路图

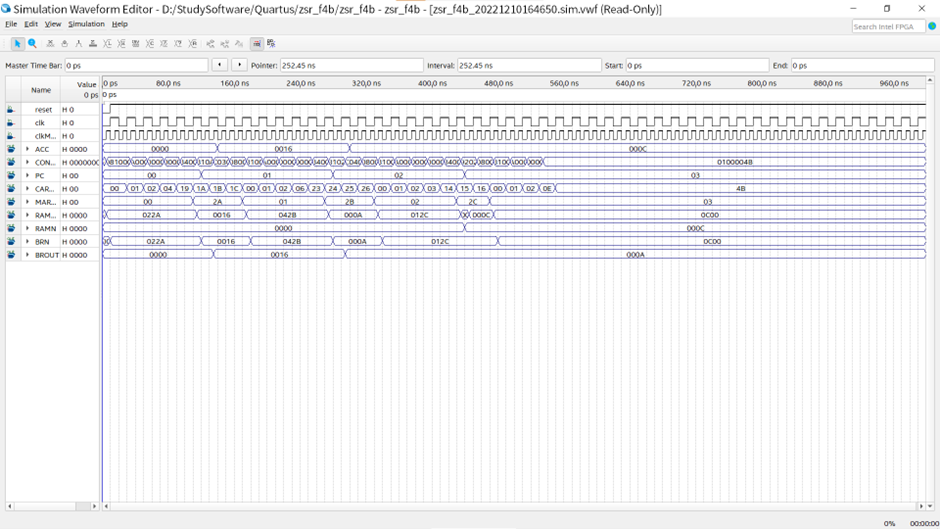




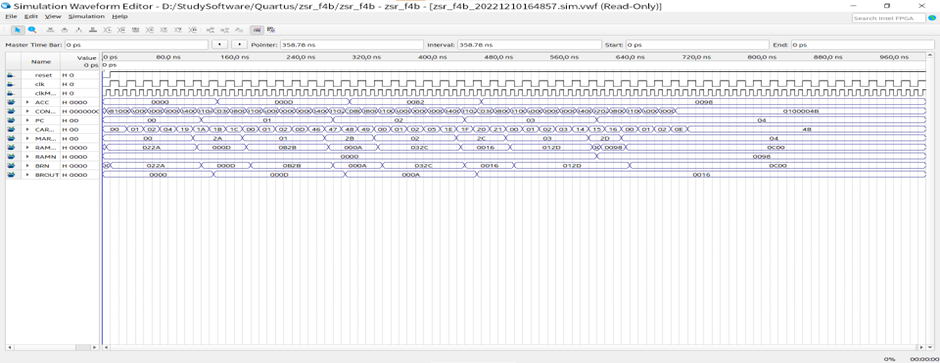
1. 波形测试

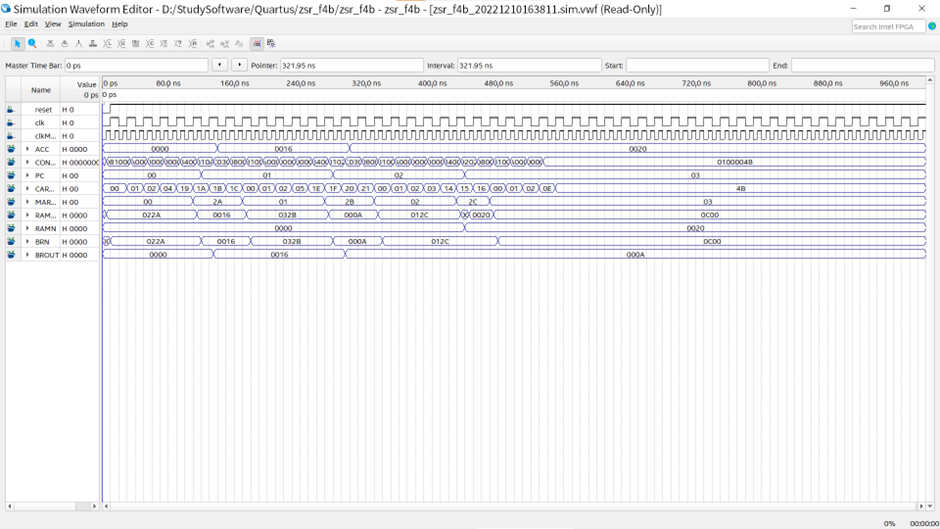
开始时PC值为0，PC将指令地址通过地址总线传递给MAR。在开始时由于CAR的值为0，然后会将CAR中的值0传入通过下图红线到rom中找出对应的微指令。然后通过下图红线数据线将微指令传入到control中。control获取指令，传递信号给MAR。MAR从通过下图红线从ram(0)中取出指令022A。再通过下数据总线传输到MBR中，其中02为操作码，2A为地址码。此时该CPU会将读信号R置为1，并且将CAR的值+1变为1，然后会将CAR中的值1像上述过程一样传入到rom中找出对应的微指令，并通过数据线将微指令传入到control 中，所以该CPU会读取MBR 中存的操作码字段通过下图红线地址总线读出并存入到OP(IR)中并且将CAR的值+1变为1。此时CAR的值为2，然后会将CAR中的值2传入到rom中找出对应的微指令，并通过数据线将微指令传入到 control中，所以该CPU会执行CAR←CAR+OP操作，使得CAR的值变为04；然后会将CAR中的值4传入到rom中 找出对应的微指令，并通过数据线将微指令传入到control中，此时CAR的值为04，所以该CPU会执行 CAR←19，使得CAR的值变为19；此时CAR的值为19，然后会将CAR中的值19传入到rom中找出对应的微指令， 并通过数据线将微指令传入到control中，至此正式进入LOAD操作，先将MBR中地址码通过下图红线地址 总线传入到MAR中

然后ACC中的值和BR中的值相加并存回ACC使得ACC值为16，最后会将CAR的值+1；此时整个LOAD操作执行 完毕，CAR的值为1c，然后会将CAR中的值1c传入到rom中找出对应的微指令，并通过数据线将微指令传入 到control中，该CPU会将CAR中的值置零。通过以上操作就完成了 0 : 022A; Load 2A这条指令。此时PC值为1，PC中的值会通过地址总线传到MAR，然后从ram(1)中取出指令032B并通过数据总 线传输到MBR中，其中03为操作码，2B为地址码。并且在开始时由于CAR的值为0，然后会将CAR中的值0传 入到rom中找出对应的微指令，并通过数据线将微指令传入到control中，此时该CPU会将读信号R置为1， 并且将CAR的值+1变为1；此时CAR的值为1，然后会将CAR中的值1传入到rom中找出对应的微指令，并通过 数据线将微指令传入到control中，所以该CPU会读取MBR中存的操作码字段通过地址总线读出并存入到OP 中并且将CAR的值+1变为1；此时CAR的值为2，然后会将CAR中的值2传入到rom中找出对应的微指令，并通过数据线将微指令传入到control中，所以该CPU会执行CAR←CAR+OP操作，使得CAR的值变为05；此时CAR 的值为05，然后会将CAR中的值5传入到rom中找出对应的微指令，并通过数据线将微指令传入到control 中，所以该CPU会执行CAR←1E，使得CAR的值变为1E；此时CAR的值为1E，然后会将CAR中的值1e传入到rom 中找出对应的微指令，并通过数据线将微指令传入到control中，至此正式进入ADD操作，先将MBR中地址 码通过地址总线传入到MAR中，然后将CAR的值+1；此时CAR的值为1f，然后会将CAR中的值1f传入到rom中 找出对应的微指令，并通过数据线将微指令传入到control中，此时CPU会执行PC+1的操作，并且将读信 号R置为1，最后将CAR的值+1；此时CAR的值为20，然后会将CAR中的值20传入到rom中找出对应的微指令，并通过数据线将微指令传入到control中，该CPU会将MBR中的操作码对应的操作数000A通过数据总线传入 到BR中，然后ACC中的值和BR中的值相加并存回ACC使得ACC值为0020，最后会将CAR的值+1；此时整个ADD 操作执行完毕，CAR的值为21，然后会将CAR中的值21传入到rom中找出对应的微指令，并通过数据线将微 指令传入到control中，该CPU会将CAR中的值置零。



指令操作过程基本和上面加法指令一致，不同点在于042B的操作码OP为04加上CAR上的02为06，此时CAR 的值为06，然后会将CAR中的值6传入到rom中找出对应的微指令，并通过数据线将微指令传入到control 中,所以该CPU会执行CAR←23，使得CAR的值变为23；此时CAR的值为23，然后会将CAR中的值23传入到rom 中找出对应的微指令，并通过数据线将微指令传入到control中,至此正式进入SUB操作，先将MBR中地址 码通过地址总线传入到MAR中，然后将CAR的值+1；此时CAR的值为24，然后会将CAR中的值24传入到rom中 找出对应的微指令，并通过数据线将微指令传入到control中,此时CPU会执行PC+1的操作，并且将读信号 R置为1，最后将CAR的值+1；此时CAR的值为25，该CPU会将MBR中的操作码对应的操作数000A通过数据总 线传入到BR中，然后ACC中的值和BR中的值相减并存回ACC使得ACC值为000C，最后会将CAR的值+1；此时 整个SUB操作执行完毕，CAR的值为26，然后会将CAR中的值26传入到rom中找出对应的微指令，并通过数 据线将微指令传入到control中,该CPU会将CAR中的值置零。





以第一个循环的ADD A3指令为例，此时PC值为05，PC的值通过地址总线传到MAR，然后从ram中取出指令03A3并通过数据总线传输到MBR中。开始时由于CAR的值为0，然后会将CAR中的值0传入到rom中找出对应的微指令，并通过数据线将微指令传入到control中，此时该CPU会将读信号R置为1，并且将CAR的值+1变为1；此时CAR的值为1，然后会将CAR中的值1传入到rom 中找出对应的微指令，并通过数据线将微指令传入到control中，所以该CPU会读取MBR中存的操作码字段 通过地址总线读出并存入到OP中并且将CAR的值+1变为1；此时CAR的值为2，然后会将CAR中的值2传入到 rom中找出对应的微指令，将微指令传入到control，所以CPU会执行CAR←CAR+OP操作，使得CAR的值变为05；此时CAR的值为05，然后会将CAR中的值5传入到rom中找出对应的微指令，并通过数 据线将微指令传入到control中，所以该CPU会执行CAR←1E，使得CAR的值变为1E；此时CAR的值为1E，然 后会将CAR中的值1e传入到rom中找出对应的微指令，并通过数据线将微指令传入到control中，至此正式 进入ADD操作，先将MBR中地址码通过地址总线传入到MAR中，然后将CAR的值+1；此时CAR的值为1f，然后会将CAR中的值1f传入到rom中找出对应的微指令，并通过数据线将微指令传入到control中，此时CPU会 执行PC+1的操作，并且将读信号R置为1，最后将CAR的值+1；此时CAR的值为20，然后会将CAR中的值20传 入到rom中找出对应的微指令，并通过数据线将微指令传入到control中，该CPU会将MBR中的操作码对应 的操作数0064通过数据总线传入到BR中，然后ACC中的值和BR中的值相加并存回ACC使得ACC值为0064，最 后会将CAR的值+1；此时整个ADD操作执行完毕，CAR的值为21，然后会将CAR中的值21传入到rom中找出对应的微指令，并通过数据线将微指令传入到control中，该CPU会将CAR中的值置零。